Lightweight Remote Procedure Call

李恒济

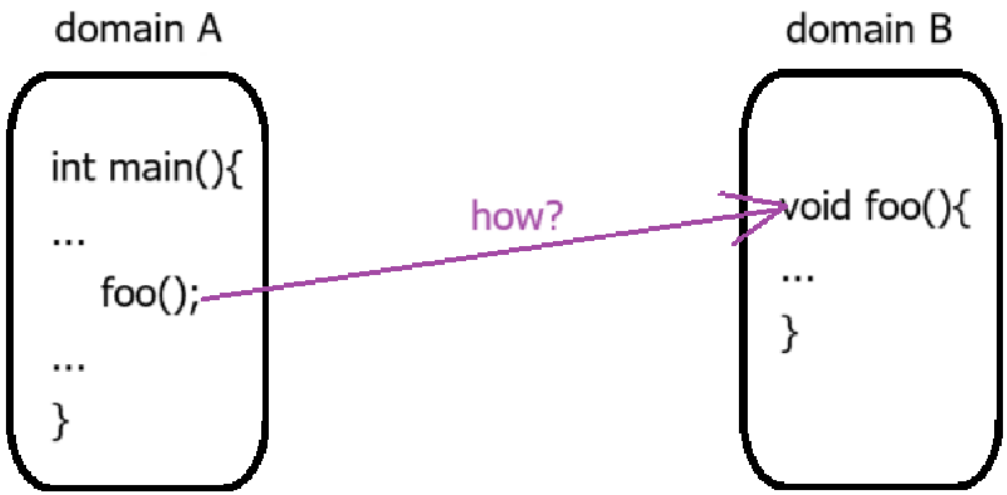
南京大学匡亚明学院

July 16, 2025

研究背景

Small-kernel 操作系统将系统的单个功能模块放置在独立的地址空间（domain）.

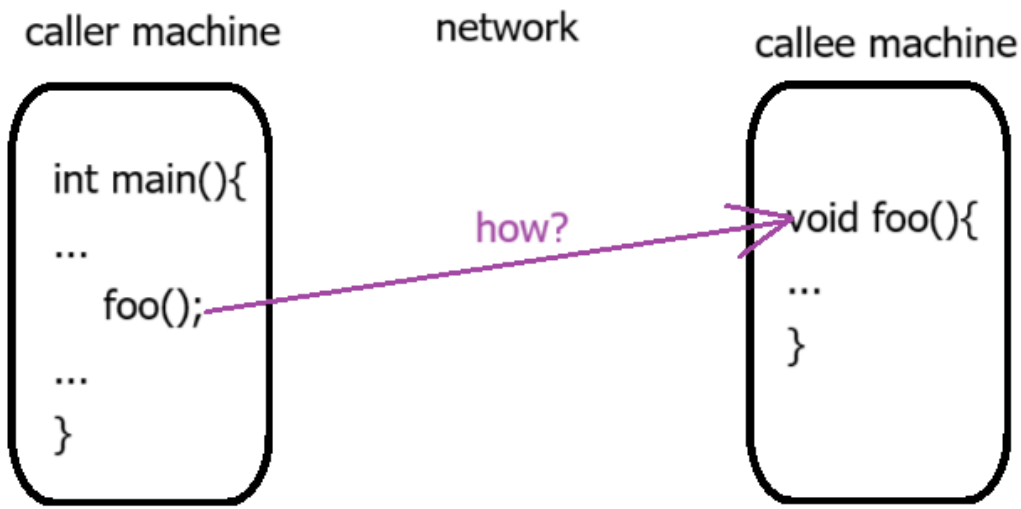
- 如何调用其他地址空间定义的过程？



研究背景

分布式计算环境中子系统分布在不同的机器上–> 如何调用另一台机器上的过程？

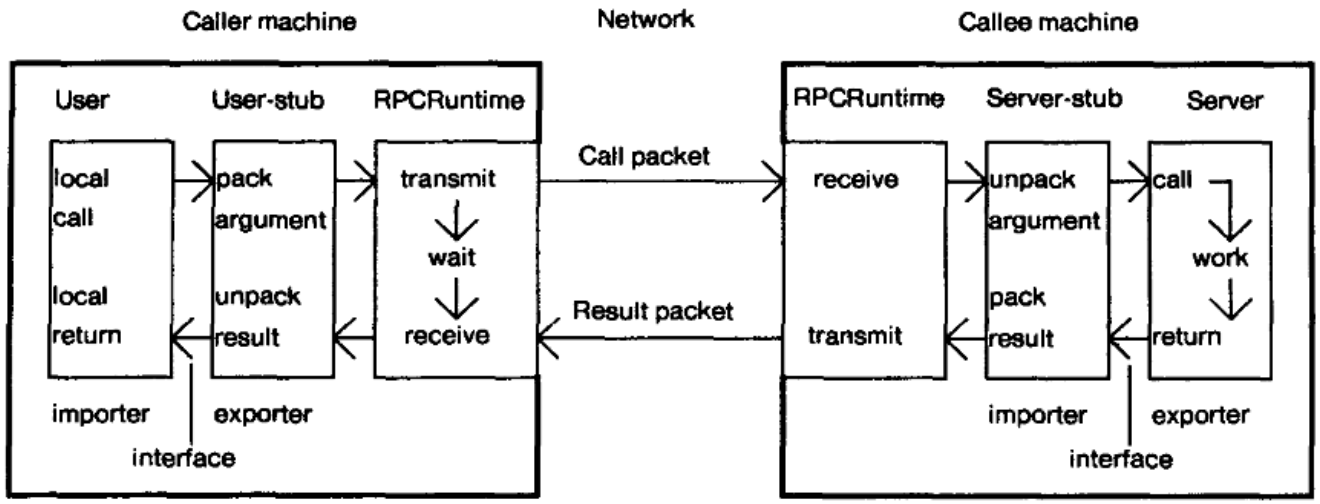
- RPC！



研究背景

RPC (Remote Procedure Call) 在分布式计算环境中取得了成功！

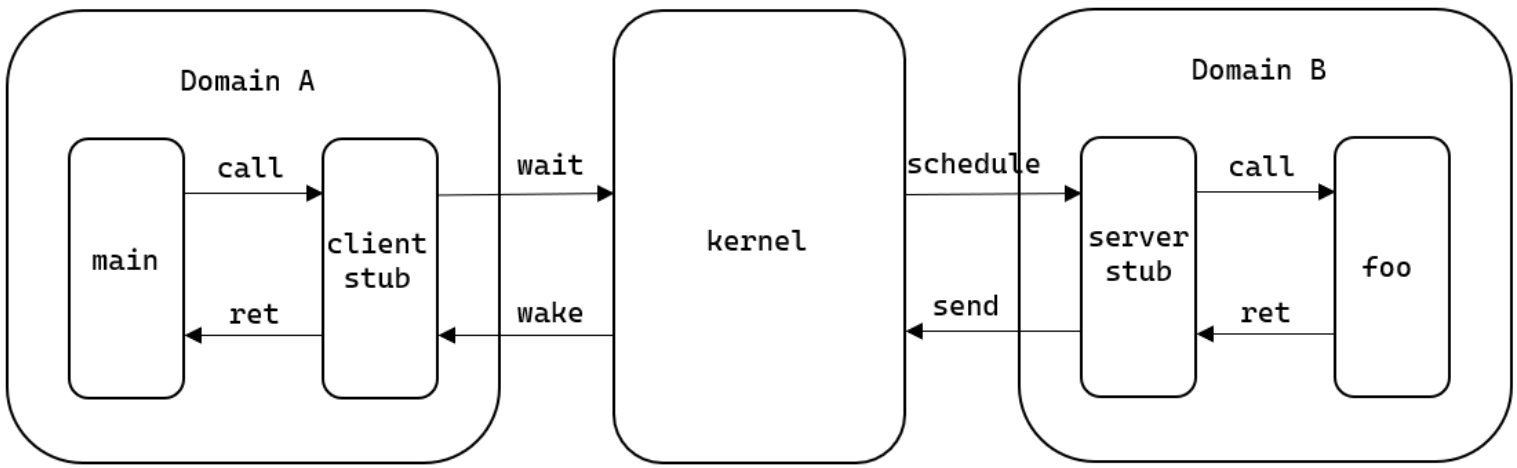
- RPC 提供了一种简单、抽象而高效的通信方式，使得开发者可以像调用本地函数一样调用远程服务器上的过程！



研究背景

Small-kernel 操作系统借鉴了 RPC 的思想，优雅的解决了跨地址空间调用的问题.

- 性能如何？



性能问题

Great for protection, bad for performance!

```
PROCEDURE Null( ); BEGIN RETURN END Null;
```

Table II. Cross-Domain Performance (times are in microseconds)

System	Processor	Null (theoretical minimum)	Null (actual)	Overhead
Accent	PERQ	444	2,300	1,856
Taos	Firefly C-VAX	109	464	355
Mach	C-VAX	90	754	664
V	68020	170	730	560
Amoeba	68020	170	800	630
DASH	68020	170	1,590	1,420

问题溯源

What's common case?

- 绝大多数调用是 cross-domain 而非 cross-machine!
- 绝大多数参数是简单的而不是复杂的!

Table I. Frequency of Remote Activity

Operating system	Percentage of operations that cross machine boundaries
V	3.0
Taos	5.3
Sun UNIX+NFS	0.6

(a)

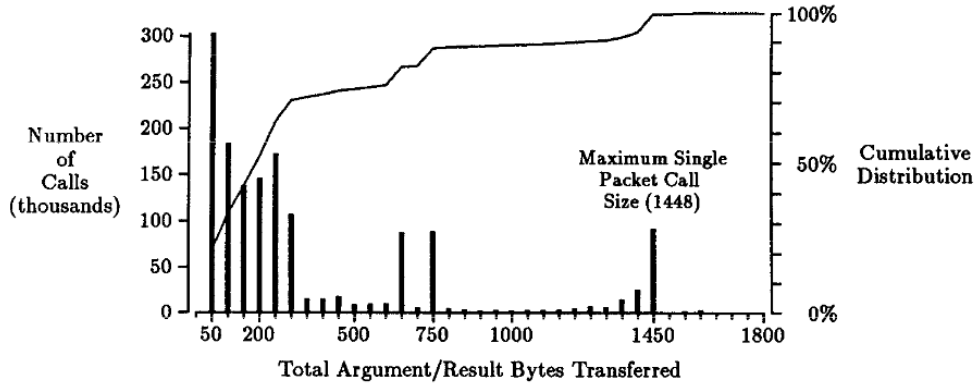


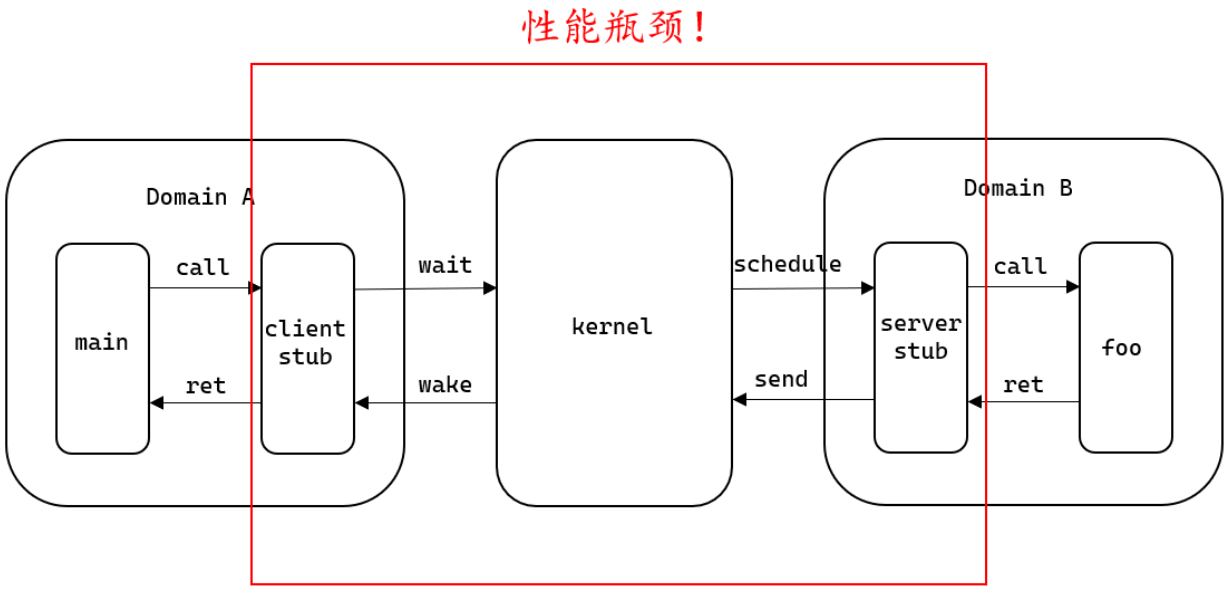
Fig. 1. RPC size distribution.

(b)

问题溯源

当前 RPC 系统中的 cross-domain 调用是基于 cross-machine 的机制实现的!

- 在不同线程间传递（可能很大、很复杂）消息.



问题根源

It violates a basic tenet of system design by failing to isolate the common case！

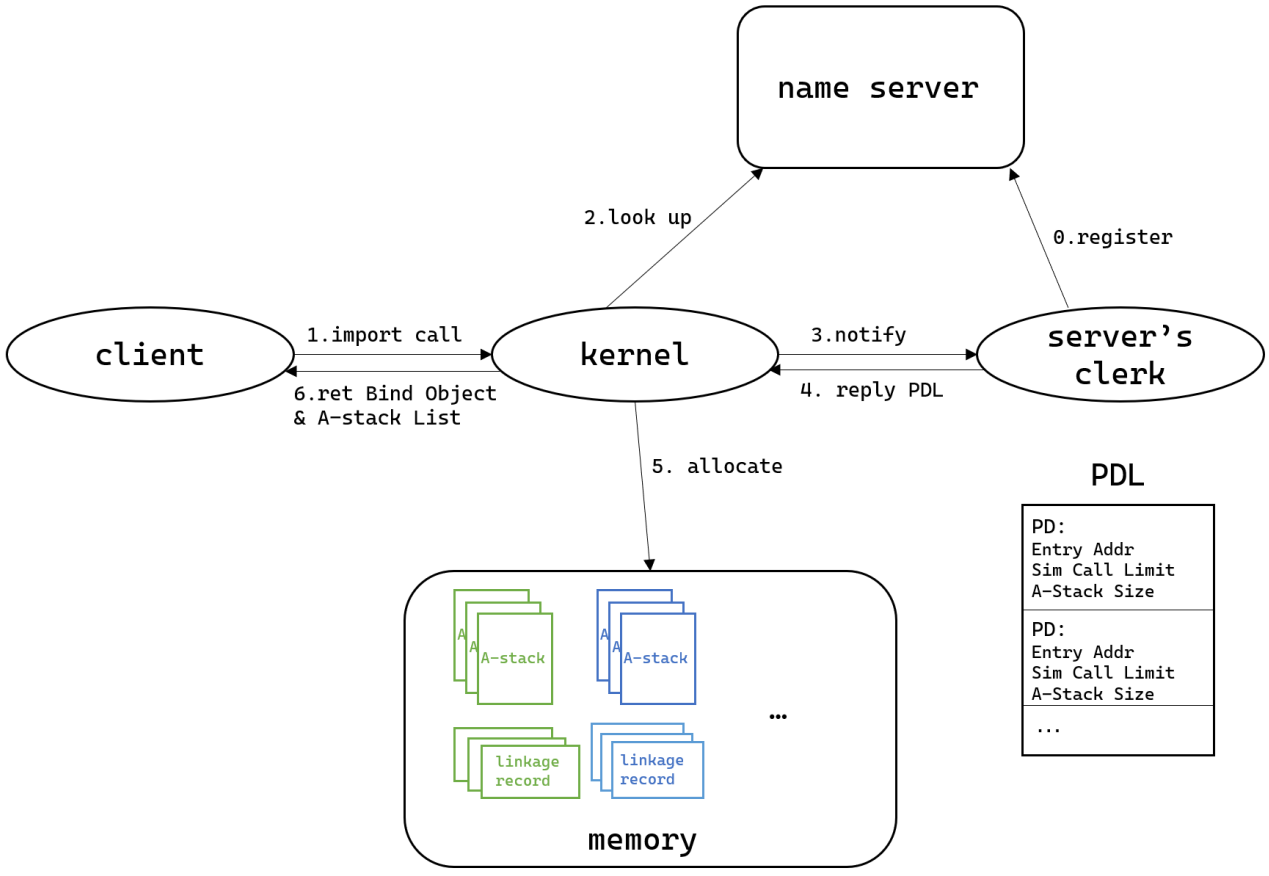
问题求解

解：为 cross-domain 调用设计 fast-path –> LRPC !

- Simple control transfer.
  - 直接让 client 去执行 server 域中被调用的过程–> 省去调度.
- Simple data transfer.
  - 使用一块共享内存（A-stack）传参–> 避免不必要的参数复制.
- Simple stubs.
  - 不再打包消息，只是压栈然后陷入内核.
- Design for concurrency.
  - 锁的粒度很细，利用多处理器加速.

Binding

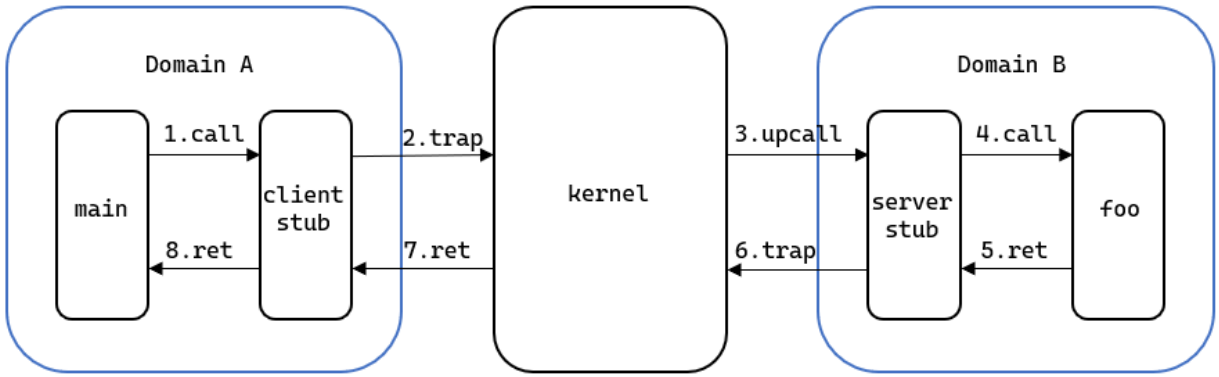
调用前先绑定（Binding），分配 A-stack 和 Binding Object.



Calling

1. client 直接调用本地的 stub 过程.
2. stub 使用 FILO 队列维护绑定时分配的 A-stacks，被调用时:

    2.1. 从队列取一个 A-stack.
    2.2. 将参数压入 A-stack.
    2.3. 将 A-stack 地址、Bind Object 以及过程标识符（procedure identifier）存入寄存器.
    2.4. 陷入内核.



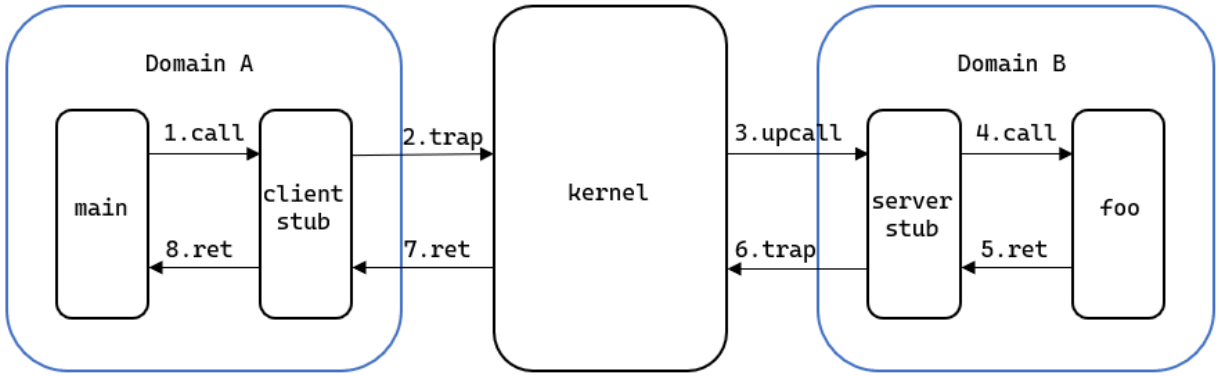
Calling

3.1. 验证 A-stack 地址、Binding Object 和过程标识符，定位 PD 和 linkage record .

3.2. 在 linkage record 记录 caller 返回地址和当前栈指针，将 linkage 压入 linkages 栈 (in TCB).

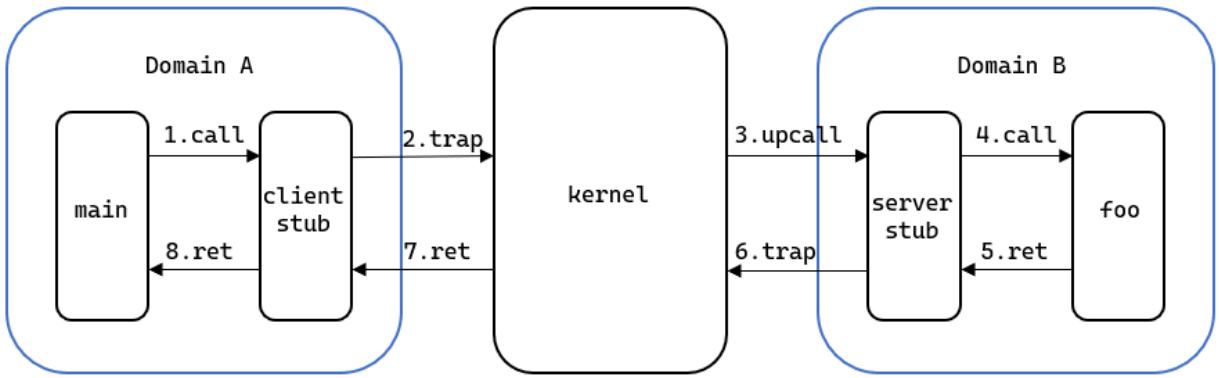
3.2. 为 server domain 分配执行栈 (E-stack)，将用户栈指针指向 E-stack.

3.3. 更改页表基地址寄存器，upcall server's stub.



Calling

4. stub 调用本地过程.
5. 过程返回，返回值被保存在 A-satck.
6. stub 陷入内核.
7. 内核更改页表基地址寄存器，回到 client's stub.
8. client's stub 返回.



更多细节

- Stub：得益于参数传递的简单性，可以自动生成简单的汇编代码 –> 可移植性?
- Slow-path : cross-machine 调用和复杂、重量级参数（如链表）–> 回归传统 RPC.
- 多处理器：
  - 只需要对 A-stack 队列加锁，并发性好.（存疑）
  - 可利用在 server 上下文中空转的处理器，减少上下文切换开销.

Advantage

Table III. Copy Operations for LRPC versus Message-Based RPC

Operation	LRPC	Message passing	Restricted message passing
Call (mutable parameters)	A	ABCE	ADE
Call (immutable parameters)	AE	ABCE	ADE
Return	F	BCF	BF

Code	*Copy operation*
A	Copy from client stack to message (or A-stack)
B	Copy from sender domain to kernel domain
C	Copy from kernel domain to receiver domain
D	Copy from sender/kernel space to receiver/kernel domain
E	Copy from message (or A-stack) into server stack
F	Copy from message (or A-stack) into client's results

Performance

Table IV. LRPC Performance of Four Tests (in microseconds)

Test	Description	LRPC/MP	LRPC	Taos
Null	The Null cross-domain call	125	157	464
Add	A procedure taking two 4-byte arguments and returning one 4-byte argument	130	164	480
BigIn	A procedure taking one 200-byte argument	173	192	539
BigInOut	A procedure taking and returning one 200-byte argument	219	227	636

Table V. Breakdown of Time (in microseconds) for Single-Processor Null LRPC

Operation	Minimum	LRPC overhead
Modula2+ procedure call	7	—
Two kernel traps	36	—
Two context switches	66	—
Stubs	—	21
Kernel transfer	—	27
Total	109	48

一些问题

A-stack 是一块共享内存并不是运行栈!

- 要求语言环境允许使用单独的参数指针而不依赖执行栈指针.

一些问题

- It is still possible for a client or server to asynchronously change the values of arguments in an A-stack once control has transferred across domains.

一些问题

- During the return from an LRPC, the client stub copies returned values from the A-stack into their final destination. **No added safety** comes from first copying these values out of the server's domain into the client's, either directly or by way of the kernel.
- 不敢苟同:(
- 我个人认为还是要给 A-stack 分配锁!

一些问题

...
For example, the Write procedure exported by a file server takes an array of bytes to be written to disk. The array itself is not interpreted by the server, which is made no more secure by an assurance that the bytes will not change during the call.

- 依旧不敢苟同:(
- 我个人认为还是要给 A-stack 分配锁!!!

参考文献

1. B. N. Bershad, E. Anderson, D. Lazowska, and H. M. Levy, “Lightweight Remote Procedure Call”.
2. D. Schroeder, “Performance of Firefly RPC,” ACM Transactions on Computer Systems, vol. 8, no. 1.
3. A. D. Birrell and B. J. Nelson, “Implementing remote procedure calls,” ACM Transactions on Computer Systems, vol. 2, no. 1, 1984.
4. W. Bell, “Lightweight Remote Procedure Call (LRPC),” lecture notes, Mar. 1999. Based on previous notes by K. LoGuidice, Mar. 1998. [Online]. Available: link [Accessed: Jul. 13, 2025].

Thanks!